HTML Kaleidoscope – how it works

The kaleidoscope is based on the Price Kaleidoscope produced by the German Statistical Office, and Statistics New Zealand expressed interest in a New Zealand version. The data used in our kaleidoscope comes from Stats NZ.

A number of concepts were used in order to get the HTML kaleidoscope functional. An HTML form element was used in conjunction with JavaScript, along with AJAX (Asynchronous JavaScript And XML). Zooming is achieved through AJAX, and tooltips are achieved with JavaScript functions that are called when onmouseover, onmousemove and onmouseout events occur. To get the HTML elements interacting with the SVG elements, I have used inline SVG which is valid using the HTML5 mark-up language.

There are 2 JavaScript files included in the document: vars.js and functions.js. The first contains all the global variables needed and is created by R by substituting information into a template file. The functions file simply contains all the functions that make the page interactive - R does not insert any information into this file.

Tooltips

There are 2 groups of elements in the kaleidoscope that we want to trigger the display of tooltips when we interact with them via the mouse. The first group is consists of the labels on the left and right sides, and the second group consists of the polygons of the kaleidoscope.

When we mouse over one of the labels for the very first time, a check is done to see whether the "label tooltip" elements exist. If not, we create these and append them to the inline SVG document via JavaScript, using the HTML DOM (Document Object Model), which describes the document as a tree-type structure with parent nodes and children nodes. The label tooltip is simply a tooltip that displays the text "Click to zoom", alerting the viewer that a zooming functionality exists and can be explored by clicking on the labels. Zooming is only available from the top level. Hovering over a label also displays a thick black border around the polygon of the corresponding group, and brings it to the front so that parts of the border are not hidden behind subsequently defined polygon elements. The effect of bringing an element to the front is achieved by cloning nodes, setting their fill to none and then appending them at the end of the inline SVG document.

A similar process occurs when we mouseover the polygons. Hovering over a polygon gives it a thick black border and brings it to the front. If it is the first time the tooltip function has been called, the tooltip elements are created and appended to the inline SVG document. Then, by making use of the argument(s) provided by the event to the called function, we can determine what category the current polygon represents. We can use this information to extract the category name, weight and price change from our global variables in vars.js. This tooltip is then updated with this information.

An onmousemove attribute calls a function that moves the tooltips as we move the mouse over the polygon. Tooltips are by default displayed on the right side of the cursor. The function which moves tooltips detects when the end of the tooltip reaches or extends beyond the edge of the inline SVG document, and repositions the tooltip onto the left side of the cursor when this happens to ensure that tooltips are never cut off.

When the mouse leaves the tooltip-producing elements, an onmouseout event calls a function that removes the cloned polygon and sets the visibility of the tooltip to "hidden".

Changing which quarters are compared

An HTML form was used to obtain this functionality. The form consists of two drop-down lists (select elements with option elements as children) and an 'Update button' which is an input element of type 'button'. Note that the option elements are not hand-coded into the document; R automates this part for us by inserting information into an HTML template file.

When the update button is pressed, the updateValues() function is called. This function 'reads' the HTML form and can see which options are selected. Then it extracts the price information for the selected quarters from the global variables in vars.js. A loop is used to run through the arrays, calculating a percentage change in price and determining which fill colour each quarter should have based on the price changes. These new price changes are then globally assigned so that they overwrite the previous information contained in that variable (so that tooltips can access this information later). Then we grab all the polygon elements using the standard getElementsByTagName() JavaScript function. This gives us an array of nodes (polygon objects) which we can loop through. We read the ID attributes because we only want to change the fill of polygons that have a certain substring in their ID. Each time we have a match we change the fill colour (this is made easier by the fact that the order of the polygons is the same as the order of information in our JavaScript variables).

The updateValues() function also recalculates price changes for the groups and updates these values (they appear underneath the labels on the sides). The heading above the legend is also updated.

There are two additional features included in the HTML form: a 'back one' link (<<) and a 'ahead one' link (>>). The back one link simply shifts both the comparison and baseline quarters back by one quarter if possible, then calls updateValues() so that we don't need to click the update button. Similarly, the ahead one link shifts the comparison and baseline quarters ahead by one quarter if possible and then calls updateValues().

Zooming

The way through which the process of updating price changes and fill colours occurs was the main reason for using AJAX for zooming. We can only keep the updated global variables if we remain on the same page. By using AJAX we can change the page content without leaving the page. AJAX allows us to send requests to the server and load the response within a JavaScript function.

When we click on one of the side labels of the top kaleidoscope our 'zoom' function is called. This function sends a request off to a file on the server called returnSvg.php. PHP (Hypertext Preprocessor) is a server-side language: PHP code is executed before the page is sent to the client who requested it. The returnSvg.php file sends back SVG code if we are zooming in. Once the response is loaded, we then replace the SVG code with the SVG code of the sub-kaleidoscope. A second request to the server loads a new vars.js file which redefines global variables, and executes that code. Because the retrieved SVG contains the default price changes and fill colours, updateValues() is called once the SVG content has been loaded.

In each sub-kaleidoscope a 'Back' link appears at the bottom. This is necessary because when we zoomed in we never left the page we started at: we simply changed the content. Using the

browser’s back button does not produce the desired effect! Clicking on the back link calls the zoom function, but this time we don’t need to load anything from the server. This is because the function that is called when we hover over a label grabs the top level SVG content and assigns it to a global variable the first time it is called. This allows us to ‘go back’ much faster than if we use the method of sending a request and waiting for a response. It is faster because the top level SVG file is over 200KB in size, and it is illogical to request and wait for this file multiple times when the content is always the same.